Anytime Inductive Logic Programming

Tony Lindgren
Department of Computer and Systems Sciences
Stockholm University and Royal Institute of Technology
Electrum 230, 164 40 Kista, Sweden

Abstract

Anytime algorithms refers to algorithms that "always" can produce a result. Often the result of the algorithm depends on the time at hand, the longer the time, the better the answer. In this paper we present an easy way of turning regular *Inductive Logic Programming* (ILP) algorithms such as *Divide-And-Conquer* (DAC) and *Separate-And-Conquer* (SAC) into anytime algorithms. We conduct experiments with these anytime algorithms and introduce a simple heuristic called *squared quota*, that we compare with an established one, *information gain*. It seems that squared quota is better suited for a small *window size* of example data, and hence better to use in anytime systems. A comparison between SAC and DAC reveals that they excel in different combinations of examples/background knowledge.

1 Introduction

The area of artificial intelligence (AI) that studies learning is called *machine learning*. According to [4] there are four main approaches to machine learning: *decision trees*, *neural networks*, *genetic algorithms* and *Inductive Logic Programming* (ILP). The benefits of using ILP is the use of background knowledge and the expressiveness of the logical language. For an introduction to ILP the reader is suggested to consult [3, 4]. If we have a data set consisting of two classes, positive and negative, then very simplified, one can regard an ILP system as one which from positive and negative examples, i.e., its training data, create rules that cover the positive examples and exclude the negative ones. These rules can then be used to classify test examples. A perfect strategy would need few learning examples and would classify all of the test examples correctly. In reality this is not the case.

The relative ineffectiveness of ILP systems caused of the machine intensive Prolog language, motivates an anytime scheme so that the entity waiting for rules can get them when it needs them and not when the algorithm has finished, which can take a considerable longer time. We will in this paper describe how one can turn regular ILP learning algorithms into anytime ILP learning algorithms.

2 Converting ILP to anytime ILP

A simple way to convert *Separate-And-Conquer* (SAC) and *Divide-And-Conquer* (DAC) into anytime SAC/DAC is to write every induced rule to a certain file. From that file we can, in a multitasking environment, ask questions towards that file, and let the learning process continue. We will only have to read in the file at certain intervals so that we always have the latest version of the rules. For a more thorough description of SAC and DAC see [1].

2.1 Heuristic used

The notion of *window size* refers to how many examples that the heuristic has at its disposal to make a decision upon. In our case a window size of 20 means that the heuristic uses 20 percent of the example set.

2.1.1 Squared quota

squared quota $= \frac{cov(R,E^{+})^2}{cov(R,E^{-})+1}$
Where $cov(R, E)$ denotes the number of examples E that is covered by R. Squared quota works as follows: The case for which we use the squared quota instead of just quota is because if we have two clauses such that clause one covers 100 positive examples and 10 negative examples, which gives us a quota of 10, and clause two covers 1000 positive examples and 100 negative we also get 10 as quota. If one reasons a little bit one sees that the second clause probably is more promising to resolve upon. Using squared quota the quota for clause one would almost be 909 and for the second clause 9900, which gives it a higher rating than

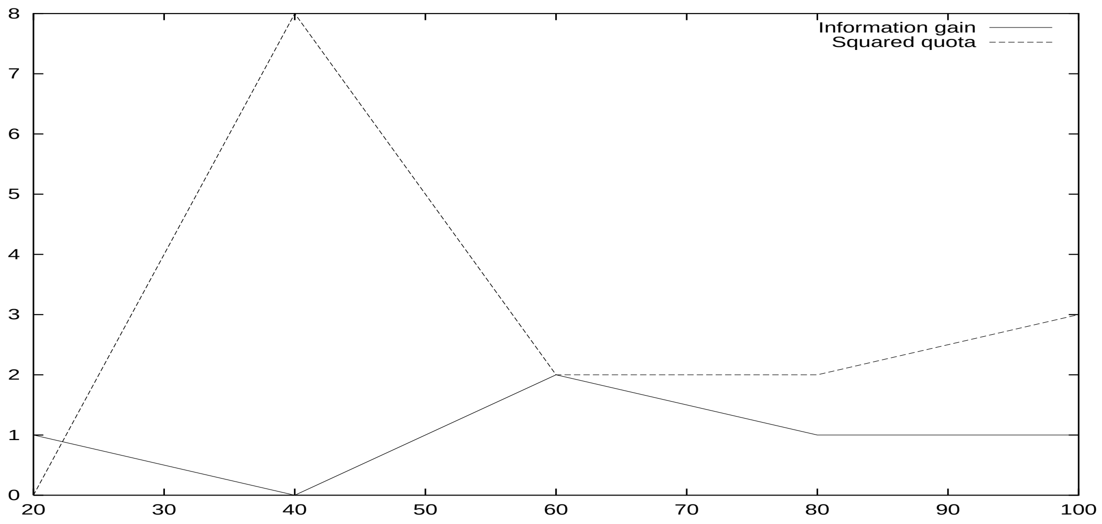

Figure 1: Amount of significant differences in accuracy between squared quota and information gain

that of clause one. The 1 is there so that we can't get zero in the denominator.

2.1.2 Information gain

information gain =
$$cov(R, E^+)\left(log_2 \frac{cov(R,E^+)}{cov(R,E^+ \cup E^-)} - log_2 \frac{cov(C,E^+)}{cov(C,E^+ \cup E^-)}\right)$$
Where $cov(R, E)$ denotes the number of examples E that is covered by R. Information gain works as follows: The entropy of a resolvent R of C, is measured with $log_2 \frac{cov(R,E^+)}{cov(R,E^+ \cup E^-)}$ that is, it measures the diversity of the examples covered by R. If R only covers positive examples then the entropy is zero. So if we have a clause C that has the entropy 1, then it covers as many negative as positive examples. Then the resolvent R of C that gets the lowest entropy is the best resolvent.

2.2 Experimental setting

Two experiments were conducted, first one to give an big picture of the performance of the algorithms and a second experiment to validate interesting results from the first experiment.

The first experiment was conducted on an Sun SPARCstation4 running Solaris operating system. Four example sets [1] were used to examine the algorithms, the example sets were randomly divided in two equal sized halves so that one half could be used for training and the other half for testing. Here the two algorithms had 0.1, 0.5, 1.0, 5.0, 10.0, 50.0, 100.0, 500.0 and 1000.0 seconds at their disposal. They were also restricted to window sizes of 20, 40, 60, 80, 100 and 120 examples. The aim of the experiments was to evaluate the performance of the different algorithms with regard to the time factor. For instance, is SAC more suitable in anytime learning than DAC? Which heuristic is best suited for anytime learning?

The second experiment was conducted on Pentium III 500 MHz running Windows NT 4.0 operating system. Using McNemar's test and ten fold cross validation as suggested by [5] as statistically valid. The aim of the experiment was to validate the differences between squared quota and information gain using the SAC algorithm. The example sets were Tic-Tac-Toe, KRKI, glass and DCG. The time at disposal were the same as those for experiment one. The window sizes was 20, 40, 60, 80 and 100 percent of the training set.

For both experiments the properties measured were correctness and number of rules produced.

3 Results

Figure 1 shows the amount of significant difference between squared quota and information gain, with different window sizes, using SAC. In figure 1 the x-axis denotes window size in percent and the y-axis number of instances that information gain respectively squared quota is significantly better than the other.

[1] All sets where taken from the UCI Machine Learning Repository. The sets are Tic-Tac-Toe, Shuttle, Promoters and King+Rook vs. King+Pawn on a7. With the Tic-Tac-Toe example set we used two different types of background knowledge, see also [1]. Hence we examined five domains.

Time (in seconds)	Domain	SAC with sq and inf, * denotes significant difference in accuracy									
		20		40		60		80		100	
		inf	sq	inf	sq	inf	sq	inf	sq	inf	sq
1	TicTacToe	6.25	5.21	4.38	7.62*	3.55	4.49*	0.0	5.21	0.0	0.0
	Num. Rules	0.2	0.22	1.0	1.0	1.0	1.0	0.0	0.2	0.0	0.0
	KRKI	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Num. Rules	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Glass	87.5	87.5	90.18	90.18	0.0	0.0	0.0	0.0	0.0	0.0
	Num. Rules	2.0	2.0	3.7	3.7	0.0	0.0	0.0	0.0	0.0	0.0
	DCG	0.0	0.0	2.65	0.0	0.0	0.0	13.53	13.53	15.04	15.93
	Num. Rules	0.0	0.0	0.1	0.0	0.0	0.0	0.7	0.7	0.1	0.2
5	TicTacToe	28.71	30.69	65.45	93.84*	29.75	34.45*	21.93	30.48*	13.47	21.09*
	Num. Rules	5.0	4.9	12.1	11.4	5.8	6.0	4.89	4.9	3.0	3.0
	KRKI	5.0	11.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Num. Rules	0.1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	88.39	90.1
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.3	4.3
	DCG	30.0	30.0	29.29	29.29	29.73	29.2	38.58	38.94	35.66	34.71
	Num. Rules	3.9	4.0	4.1	4.2	4.1	4.11	7.9	7.9	0.0	0.0
10	TicTacToe	99.27	99.48	95.82	98.02*	99.37*	94.68	70.77	95.51*	40.71	58.77*
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	11.1	11.3	7.9	8.6
	KRKI	21.4	21.5	11.67	12.2	0.0	0.0	0.0	0.0	0.0	0.0
	Num. Rules	2.0	2.0	0.9	10.0	0.0	0.0	0.0	0.0	0.0	0.0
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	52.74*	47.52	47.61	45.22	47.79	48.67	88.32	89.2	99.73	99.73
	Num. Rules	14.8	13.9	14.7	14.1	14.8	15.0	15.3	15.8	16.0	16.0
50	TicTacToe	99.27	99.48	95.82	98.02*	99.37	99.58	99.16	99.37	99.16	99.58
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	13.3	11.5	13.0	11.4
	KRKI	99.1	98.7	99.2	99.2	32.2*	30.3	15.8	15.9	14.8	15.5*
	Num. Rules	5.2	5.2	7.0	7.1	4.0	3.9	2.0	2.1	1.9	2.0
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	99.03	99.03	98.41	98.76	98.23	98.5	99.29	99.65	99.73	99.73
	Num. Rules	19.9	19.5	20.2	20.6	19.6	19.7	17.4	17.0	16.0	16.0
100	TicTacToe	99.27	99.48	95.82	98.02*	99.37	99.58	99.16	99.37	99.16	99.58
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	13.3	11.5	13.0	11.4
	KRKI	99.1	98.7	99.2	99.2	99.2	99.3	38.4*	31.8	28.4*	19.6
	Num. Rules	5.2	5.2	7.0	7.1	7.4	7.9	6.3	6.0	3.7	4.1
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	99.03	99.03	98.41	98.76	98.23	98.5	99.29	99.65	99.73	99.73
	Num. Rules	19.9	19.5	20.2	20.6	19.6	19.7	17.4	17.0	16.0	16.0
500	TicTacToe	99.27	99.48	95.82	98.02*	99.37	99.58	99.16	99.37	99.16	99.58
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	13.3	11.5	13.0	11.4
	KRKI	99.1	98.7	99.2	99.2	99.2	99.3	98.9	99.1	99.3	99.3
	Num. Rules	5.2	5.2	7.0	7.1	7.4	7.9	10.2	10.4	10.8	10.8
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	99.03	99.03	98.41	98.76	98.23	98.5	99.29	99.65	99.73	99.73
	Num. Rules	19.9	19.5	20.2	20.6	19.6	19.7	17.4	17.0	16.0	16.0
1000	TicTacToe	99.27	99.48	95.82	98.02*	99.37	99.58	99.16	99.37	99.16	99.58
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	13.3	11.5	13.0	11.4
	KRKI	99.1	98.7	7.0	7.1	99.2	99.3	98.9	99.1	99.3	99.3
	Num. Rules	5.2	5.2	16.3	12.2	7.4	7.9	10.2	10.4	10.8	10.8
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	99.03	99.03	98.41	98.76	98.23	98.5	99.29	99.65	99.73	99.73
	Num. Rules	19.9	19.5	20.2	20.6	19.6	19.7	17.4	17.0	16.0	16.0
inf.	TicTacToe	99.27	99.48	95.82	98.02*	99.37	99.58	99.16	99.37	99.16	99.58
	Num. Rules	12.4	11.6	16.3	12.2	14.6	13.0	13.3	11.5	13.0	11.4
	KRKI	99.1	98.7	99.2	99.2	99.2	99.3	98.9	99.1	99.3	99.3
	Num. Rules	5.2	5.2	7.0	7.1	7.4	7.9	10.2	10.4	10.8	10.8
	Glass	87.5	87.5	90.18	90.18	92.86	93.75	93.75	92.86	91.07	91.07
	Num. Rules	2.0	2.0	3.7	3.7	3.8	3.8	4.1	4.1	4.7	4.8
	DCG	99.03	99.03	98.41	98.76	98.23	98.5	99.12	99.47	99.73	99.73
	Num. Rules	19.9	19.5	20.2	20.6	19.6	19.7	17.7	17.4	16.0	16.0

Figure 2: Data from the second experiment

3.1 DAC versus SAC

In our first experiments, DAC has better accuracy than SAC in two domains (Promoters and King+Rook vs. King+Pawn on a7), is equal to SAC in one domain (Shuttle) and has lower accuracy in two domains (both Tic-Tac-Toe domains). Therefore, if one only considers accuracy it's a dead race. But if one considers the number of rules produced, then SAC is better or equal in all domains. In an comparison of their speed, both approaches have almost similar performance. DAC produces more rules in a shorter time as it, in contrast to SAC, doesn't need to restart from the overly general clause when it has produced a rule. This is often a problem though, referred to as the replication problem see [2]. The differences inherited in how the algorithm induces rules mean that they are suited to different types of data sets.

3.2 Squared quota versus information gain

In our second experiment we tried to focus on the differences caused by squared quota and information gain. In figure 2 the accuracy of the two heuristics is shown along with the number of rules produced. In figure 1 we have summarized the amount of significant differences between squared quota and information gain. Squared quota has a significant better total accuracy of 15 in the four domains while information gain has a total of 5. The results of the experiments reveal that squared quota has a small advantage over information gain, in accuracy, speed and in number of rules produced.

3.3 Window size

A small window size reduces the accuracy of the rules that is produced, but generally the learning times shortens. So if one have a limited time at disposal and a relative big data set, it may well be better to use a smaller window size than all the available training examples. For example if we look at figure 2 for the tic-tac-toe domain with a time of 10 seconds; if we use all available examples the accuracy is 40.71 percent for information gain and if we only use 20 percent of the training examples the accuracy raises to 99.27 percent, quite a difference.

4 Concluding remarks

We have introduced a new heuristic called squared quota, which seems to perform better, or at least as well, as information gain. In the comparison of SAC and DAC as to their suitability as an anytime learner there is no real winner, both perform with almost similar speed. In accuracy the story is the same: in a particular domain one approach can be better than the other but as a whole they perform equally good. The only difference that is constant is that DAC always produces more or at least equally many rules as SAC.

One interesting thing to investigate further is also the heuristic of squared quota and how well it stands to more established heuristics, especially with larger amounts of data sets.

More advanced types of anytime ILP strategies should also be considered; for example it may well be possible for the learning algorithm to adapt its learning parameters, i.e. window size, heuristic, strategy, with regard to previous experience.

References

[1] Boström H. Covering vs. divide-and-conquer for top-down induction of logic programs. *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[2] Boström H. and Asker L. Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, 1999.

[3] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 1994.

[4] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming.* Springer-Verlag, 1997.

[5] Dietterich G. Thomas. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 1998.